The System.Console class

The "Console" is a class in "System" namespace, which provided a set of properties and methods to perform I/O operations in Console Applications (Command-Prompt Applications).
It is a static class. So all the members of "Console" class are accessible without creating any object for the "Console" class.
The "Console" class is a part of BCL (Base Class Library).

Members of 'Console' class

void Write(value):
It receives a value as parameter and displays the same value in Console (Command-Prompt window).

void WriteLine(value):

It receives a value as parameter and displays the same value in Console and also moves the cursor to the next line, after the value.

void ReadKey( ):
It waits until the user presses any key on the keyboard.
It makes the console window wait for user's input.

void Clear( ):
It clears (make empty) the console window.
After clearing the screen, you can display output again, using Write( ) or WriteLine( ) methods.

string ReadLine( ):
It accepts a string value from keyboard (entered by user) and returns the same
It always returns the value in "string" type only.
Even numbers (digits) are treated as strings.

What is Variable?

Variable is a named memory location in RAM, to store a particular type of value, during the program execution.
All Variables will be stored in Stack.›For every method call, a new "Stack" will be created.
The variable's value can be changed any no. of times.
The variables must be declared before its usage.
The variables must be initialized before reading its value.
Variable's data type should be specified while declaring the variable; it can't be changed later.
The stack (along with its variables) will be deleted automatically, at the end of method execution.

How to create Variables?

Syntax to create a variable:
data_type variable_name;
[or]
data_type variable_name = value;

Set value into the variable:
variable_name = value;

Get value from the variable:
variable_name

Variable / Identifier Naming Rules

1. Variable name should not contain spaces.
Student Name: wrong
StudentName: correct

2. Variable name should not have special characters [except underscore]
Student#Name: wrong
StudentName: correct

3. Duplicate variable names are not allowed.
int x;
double x: wrong (already there was a variable with same name (x)).

4. Variable names can't be same as keywords
int void: wrong
int StudentNo: correct

What is 'type'?

'Type' specifies what type of value to be stored in memory.
"Type" is a.k.a. "data type".
Eg: int, string etc.

Classification of Types

Primitive Types:
(sbyte, byte, short, ushort, int, uint, long, ulong, float, double, decimal, char, bool)
• Strictly stores single value.

• Primitive Types are basic building blocks of non-primitive types.

Non-Primitive Types:
(string, Classes, Interfaces, Structures, Enumerations)
• Stores one or more values.
• Usually contains multiple members.

Primitive Types

sbyte:
• 8-bit signed integer
• Size: 1 byte
• Range: -128 to 127
• Default value: 0
• MinValue Command: sbyte.MinValue
• MaxValue Command: sbyte.MaxValue

byte:
• 8-bit un-signed integer
• Size: 1 byte
• Range: 0 to 255
• Default value: 0
• MinValue Command: byte.MinValue
• MaxValue Command: byte.MaxValue

short:
• 16-bit signed integer
• Size: 2 bytes
• Range: -32,768 to 32,767
• Default value: 0
• MinValue Command: short.MinValue
• MaxValue Command: short.MaxValue

ushort:
• 16-bit un-signed integer
• Size: 2 bytes
• Range: 0 to 65,535
• Default value: 0
• MinValue Command: ushort.MinValue
• MaxValue Command: ushort.MaxValue

int:
• 32-bit signed integer
• Size: 4 bytes
• Range: -2,147,483,648 to 2,147,483,647
• Default value: 0
• MinValue Command: int.MinValue
• MaxValue Command: int.MaxValue
• By default, integer literals between -2,147,483,648 to 2,147,483,647 are treated as "int" data

type.

uint:
• 32-bit un-signed integer
• Size: 4 bytes
• Range: 0 to 4,294,967,295
• Default value: 0
• MinValue Command: uint.MinValue
• MaxValue Command: uint.MaxValue
• By default, integer literals between 2,147,483,648 to 4,294,967,295 are treated as "uint" data type.

long:
• 64-bit signed integer
• Size: 8 bytes
• Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
• Range: -263 to 263 (power -1)
• Default value: 0
• MinValue Command: long.MinValue
• MaxValue Command: long.MaxValue
• By default, integer literals between 4,294,967,296 and 9,223,372,036,854,775,807 are treated as "long" data type.

ulong:
• 64-bit un-signed integer
• Size: 8 bytes
• Range: 0 to 18,446,744,073,709,551,615
• Default value: 0
• MinValue Command: ulong.MinValue
• MaxValue Command: ulong.MaxValue
• By default, integer literals between 9,223,372,036,854,775,808 and 18,446,744,073,709,551,615 are treated as "ulong" data type.

float:
• 32-bit signed floating-point number
• Size: 4 bytes
• Range: -340282300000000000000000000000000000000 to 340282300000000000000000000000000000000
• Range: -3.402823E+38 to 3.402823E+38
• Range: "MINUS three hundred fourty two hundred eighty-two three hundred nonillion" *to* "three hundred fourty two hundred eighty-two three hundred NONILLION"
• Precision: 7 digits
• Default value: 0F
• MinValue Command: float.MinValue
• MaxValue Command: float.MaxValue

double:
• 64-bit signed floating-point number
• Size: 8 bytes
• 179769313486232000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
00000 (294 0's)
to
1797693134862320000000000000000000000000000000000000000000000000000000000000
000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000 (294 0's)

- Range: -1.79769313486232E+308 to 1.79769313486232E+308
- "MINUS one hundred seventy-nine trillion seven hundred sixty-nine billion three hundred thirteen million four hundred eighty-six thousand two hundred thirty-two UNTRIGINTILLION DUOTRIGINTILLION DUOTRIGINTILLION" *to* "one hundred seventy-nine trillion seven hundred sixty-nine billion three hundred thirteen million four hundred eighty-six thousand two hundred thirty-two UNTRIGINTILLION DUOTRIGINTILLION DUOTRIGINTILLION"
- Precision: 15 digits
- Default value: 0D
- Min and Max: double.MinValue, double.MaxValue
- By default, floating-point literals in the specified range are treated as "double" data type.

decimal:

- 128-bit signed floating-point number
- Size: 16 bytes
- Range: -79228162514264337593543950335 to 79228162514264337593543950335
- "MINUS seventy-nine octillion two hundred twenty-eight septillion one hundred sixty-two sextillion five hundred fourteen quintillion two hundred sixty-four quadrillion three hundred thirty-seven trillion five hundred ninety-three billion five hundred fourty-three million nine hundred fifty thousand three hundred thirty-five" *to* "seventy-nine octillion two hundred twenty-eight septillion one hundred sixty-two sextillion five hundred fourteen quintillion two hundred sixty-four quadrillion three hundred thirty-seven trillion five hundred ninety-three billion five hundred fourty-three million nine hundred fifty thousand three hundred thirty-five"
- Precision: 28 digits
- Default value: 0M
- Min and Max: double.MinValue, double.MaxValue

char:

- 16-bit Single Unicode character
- Character literal should be written in single quotes only. Ex: 'A'
- Size: 2 bytes
- Range: 0 to 137,994 (Unicode codes that represent characters)
- Unicode is superset of ASCII.
- ASCII = 0 to 255 (English language characters only)
- Unicode = ASCII + Other natural language characters
- Default value: \0

Important ASCII / Unicode numbers for characters:

65 to 90 : A-Z
97 to 122: a-z
48 to 57: 0-9
32: Space

8: Backspace
13: Enter

string:
• Collection of Unicode characters
• String literal should be written in double quotes only. Ex: "Abc123"
• Size: Length * 2 bytes
• Range: 0 to 2 billion characters
• Default value: null

bool:
• Stores logical value (true / false)
• Possible values: true, false
• Size: 1 bit
• Default value: false

Default Literals
You can get the default value of respective type using the following syntax.
default(type)
Example: default(int) = 0

Operators

Operator is a symbol to perform operation.
Operator receives one or more operands (values) and returns one value.
Eg: +, -, *, /, == etc.

1. Arithmetical Operators
2. Assignment Operators
3. Increment and Decrement Operators
4. Comparison Operators
5. Logical Operators
6. Concatenation Operator
7. Ternary Operator

Arithmetical Operators
Used to perform arithmetical operations on the numbers
+ Addition
- Subtraction
* Multiplication
/ Division
% Remainder

Assignment Operators
Used to perform arithmetical operations on the numbers
= Assigns to
+= Add and Assigns to
-= Subtract and Assigns to

*= Multiply and Assigns to
/= Divide and Assigns to
%= Remainder Assigns to

Increment / Decrement Operators
Used to perform arithmetical operations on the numbers
It returns the incremented / decremented value and also overwrites the value of variable.

n++ Post-Incrementation (First it returns value; then increments)
++ n Pre-Incrementation (First it increments value; then returns)
n-- Post-Decrementation (First it returns value; then decrements)
--n Pre-Decrementation (First it decrements value; then returns)

Comparison Operators
Used to compare two values and return true / false, based on the condition.
== equal to
!= not equal to
< less than
> greater than
<= less than or equal to
>= greater than or equal to

Logical Operators
Checks both operands (Boolean) and returns true / false.
& Logical And (Both operands should be true). Evaluates both operands, even if left-hand operand returns false.
&& Conditional And (Both operands should be true). Doesn't evaluate right-hand operand, if left-hand operand returns false.
| Logical Or (At least any one operand should be true). Evaluates both operands, even if left-hand operand returns true.
|| Conditional Or (At least any one operand should be true). Doesn't evaluate right-hand operand, if left-hand operand returns true.

Comparison Operators
^ Logical Exclusive Or - XOR (Any one operand only should be true). Evaluates both operands.
! Negation (true becomes false; False becomes true)

Concatenation Operator
Attaches second operand string at the end of first operand string and returns the combined string.
+ "string1" + "string2" returns "string1string2" (as string)
"string" + number returns "stringnumber" (as string)
number + "string1" returns "numberstring" (as string)

Ternary Conditional Operator
It evaluates the given Boolean value;
Returns first expression (consequent) if true;
Returns second expression (alternative) if false.
? : (condition)? consequent : alternative

Operator Precedence

Control Statements

Conditional Control Statements:
- if (simple-if, if-else, else-if, nested-if)
- switch-Case

Looping Control Statements:
- while
- do-While
- for

Jumping Control Statements:
- goto
- break
- continue

simple-if

Simple-if - Syntax
```
if (condition)
{
  true block here
}
```

Simple-if - Example
```
if (x < 10)
{
  System.Console.WriteLine("x is smaller than 10");
}
```

if-else

if - else - Syntax
```
if (condition)
{
  true block here
}
else
{
  false block here
}
```

if-else - Example

```
if (x > 10)
{
  System.Console.WriteLine("x is larger than 10");
}
else
{
  System.Console.WriteLine("x is smaller than or equal to 10");
}
```

else-if

else-if - Syntax

```
if (condition1)
{
  true block 1 here
}

else if (condition2)
{
  true block 2 here
}

else
{
  false block here
}
```

else If - Example

```
if (a > 10)
{
  System.Console.WriteLine("a is greater than 10");
}
else if (a < 10)
{
  System.Console.WriteLine("a is less than 10");
}
else
{
  System.Console.WriteLine("a is equal to 10");
}
```

nested-if

nested-if - Syntax

```
if (condition1)
```

```
{
  if (condition2)
  {
    true block here
  }
  else
  {
    false block 2 here
  }
}
else
{
  false block 1 here
}
```

nested If - Example

```
if (a >= 10)
{
  if (a > 10)
  {
    System.Console.WriteLine("a is greater than 10");
  }
  else
  {
    System.Console.WriteLine("a is equal to 10");
  }
}
else
{
  System.Console.WriteLine("a is less than 10");
}
```

switch-case

switch-case - Syntax

```
switch (variable)
{
  case value1: statement1; break;
  case value2: statement2; break;
  case value3: statement3; break;
  …
  default: statement; break;
}
```

switch-case - Example

```
switch ( x )
{
```

```
  case 1: System.Console.WriteLine("one"); break;
  case 2: System.Console.WriteLine("two"); break;
  case 3: System.Console.WriteLine("three"); break;
  default: System.Console.WriteLine("none"); break;
}
```

Used to check a variable value, many times, whether it matches with any one of the list of values.
Among all cases, only one will execute.
If all cases are not matched, it executes the "default case".

while

while - Syntax

```
initialization;
while (condition)
{
  while block
  incr / decr here
}
```

while - Example

```
int i = 1;
while ( i <= 10 )
{
  System.Console.WriteLine( i );
  i++;
}
```

Used to execute a set of statements, as long as the condition is TRUE.
Once the condition is false, it will exit from the while loop.

do-while

do-while - Syntax

```
initialization;
do
{
  do-while block
  incr / decr here
} while (condition);
```

do-while - Example

```
int i = 1;
do
{
System.Console.WriteLine( i ); i++;
} while ( i <= 10 );
```

Used to execute a set of statements, as long as the condition is TRUE.›Once the condition is

false, it will exit from the while loop.
It is same as "While loop"; but the difference is:
It executes at least one time even though the condition is false, because it doesn't check the condition for the first time.
Second time onwards, it is same as "while" loop.

for

for - Syntax

```
for (initialization; condition; incrementation)
{
  for block
}
```

for - Example

```
for (int i = 1; i <= 10; i++)
{
  System.Console.WriteLine( i );
}
```

Used to execute a set of statements, as long as the condition is TRUE.
Once the condition is false, it will exit from the while loop.
It is same as "While loop"; but the difference is:
We can write all loop details (initialization, condition, incrementation), in-one-line.

break

```
break - Syntax
for (initialization; condition1; incrementation)
{
  if (condition2)
  {
    break;
  }
  for block code here
}
```

break - Syntax

```
for (initialization; condition1; incrementation)
{
  if (condition2)
  {
    break;
  }
  for block code here
}
```

break - Example

```
for (int i = 0; i <= 10; i++)
{
  if (i == 6)
  {
    break;
  }
  System.Console.WriteLine(i);
}
//Output: 0, 1, 2, 3, 4, 5
```

Used to stop the execution of current loop.
It is recommended to keep the "break" statement, inside "if" statement.
It can be used in any type of loop (while, do-while, for).

continue

```
for (initialization; condition1; incrementation)
{
  if (condition2)
  {
    continue;
  }
  for block code here
}
```

continue - Syntax

```
for (initialization; condition1; incrementation)
{
  if (condition2)
  {
    continue;
  }
  for block code here
}
```

continue - Example

```
for (int i = 0; i <= 10; i++)
{
  if (i == 6)
  {
    continue;
  }
  System.Console.WriteLine(i);
}
//Output: 0, 1, 2, 3, 4, 5, 7, 8, 9, 10
```

Used to skip the execution of current iteration; and jump to the next iteration.
It is recommended to keep the "continue" statement, inside "if" statement.
It can be used in any type of loop (while, do-while, for).

nested-for

nested for - Syntax

```
for (initialization; condition1; incrementation)
{
  for (initialization; condition2; incrementation)
  {
    inner-loop code here
  }
  outer-loop code here
}
```

nested for - Example

```
for (int i = 0; i < 5; i++)
{
  for (int j = 0; j < 5; j++)
  {
    System.Console.WriteLine( j );
  }
}

//Output: 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4,
0, 1, 2, 3, 4,
```

goto

goto - Syntax

Used to jump to the specific label.
You must create a label with some specific name.
The label can be present at the top of "goto statement"; or at the bottom; but it should be in the same method.